Amazon Bedrock AgentCore

GETTING STARTED – HANDS ON

FRANK KANE

SUNDOG EDUCATION

What is AgentCore?

Hands deployment and operation of AI agents at scale

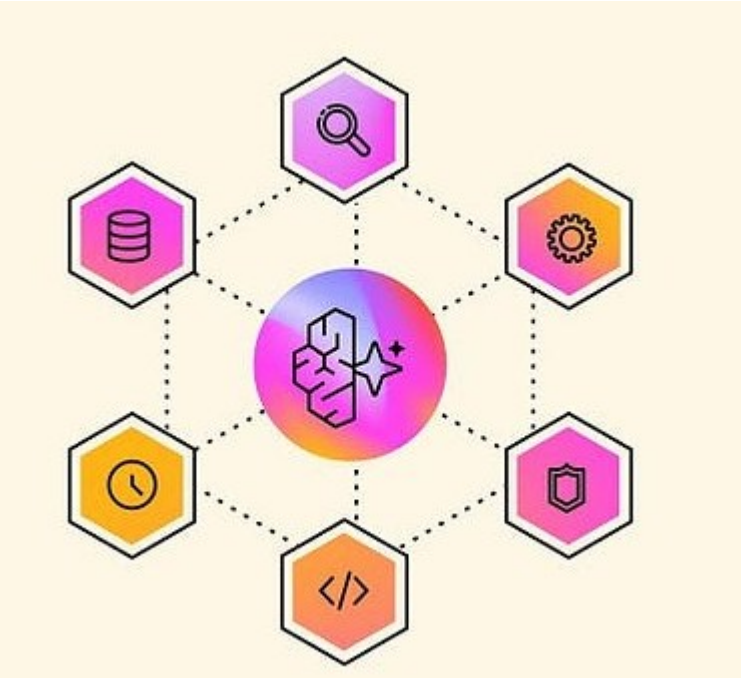
- No fussing with Docker, ECR, ECS, etc
- Serverless

And it works with ANY agent framework

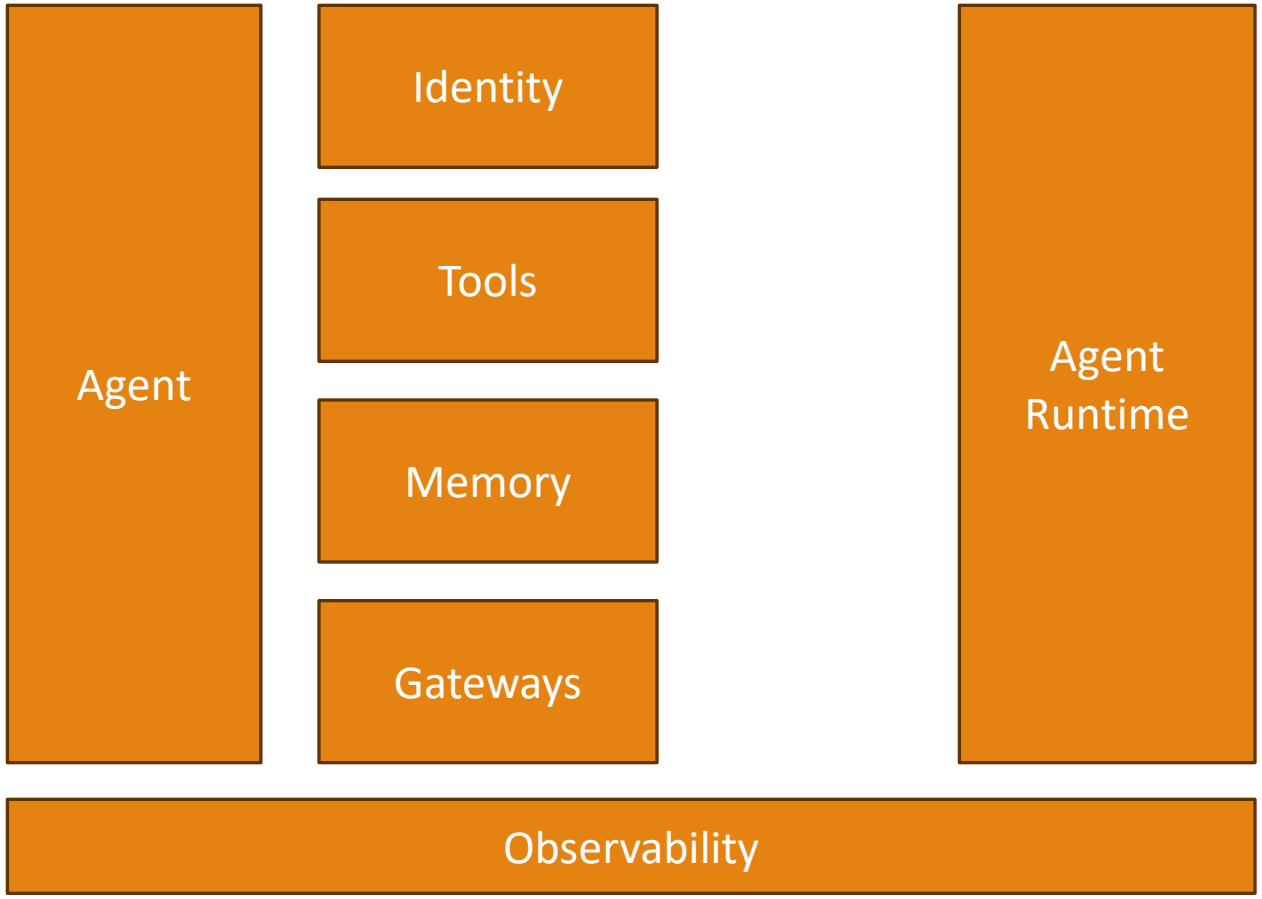
- You aren't tied to Bedrock agents
- You aren't even tied to AWS's Strands framework
- OpenAI Agent SDK, LangGraph / LangChain, CrewAI, whatever
- (although… Strands does get better support…)

Includes a "starter toolkit" to make deployment of agents to AWS super easy

Includes several tools and capabilities you can include in your agents



AgentCore Capabilities



Let’s start by building an OpenAI Agents SDK system

Our playbook

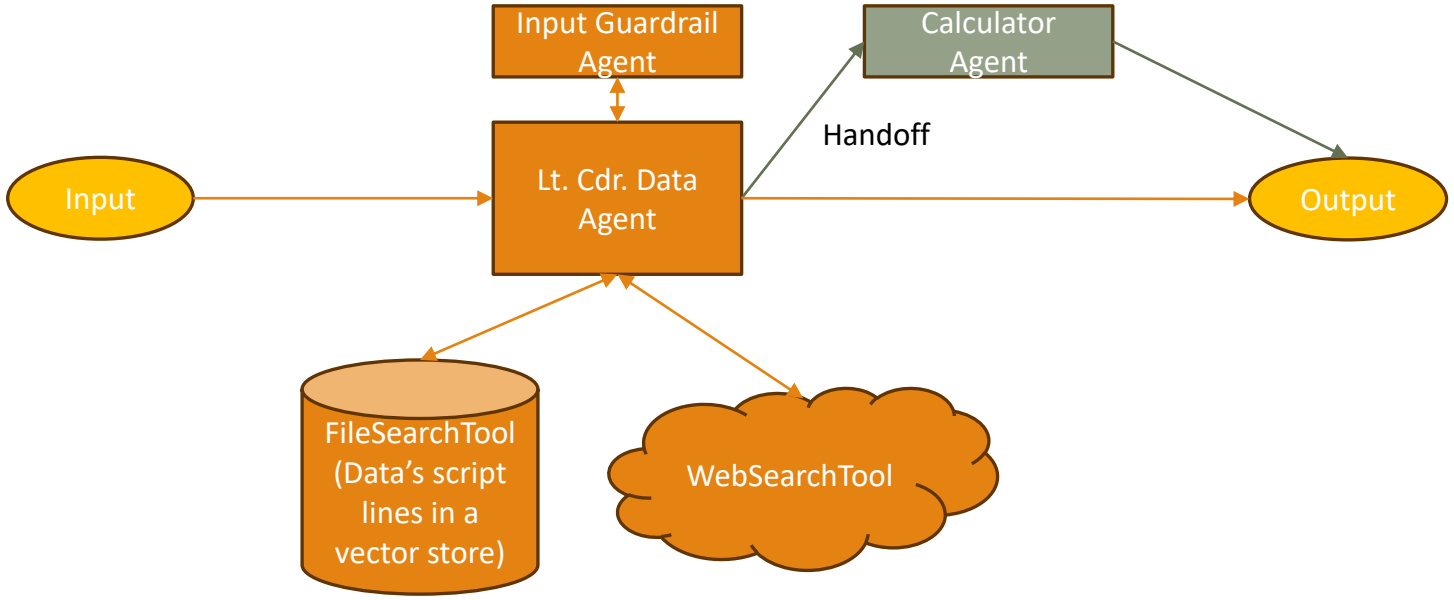
Install the AWS CLI

- Configure it for an IAM user (not root!)

Install openai and openai-agents packages for Python

Set OPENAI_API_KEY environment variable

Create a simulated “Commander Data” from TV!



AgentCore Agent Runtime

Serverless endpoints

Deploy your agent to ECR, enhanced with AgentCore capabilities

- “Starter toolkit” manages it all for you, using CodeBuild
- But you can build your own Docker containers if you want

Can have multiple endpoints

Observability dashboard lets you track usage and performance

- Through “GenAI Observability” feature in CloudWatch

Integration is super easy

File Edit Selection View Go Run ···

requirements.txt | data_agent_agentcore_memory_interpreter.py 8 | Release Notes: 1.104.0 | SetupCognito

E: > Frank.kane Dropbox > Frank Kane > AgentCore > openai-agents > data_agent_agentcore_memory_interpreter.py > ...

```python
    tools=[web_search, file_search, execute_python],   # <-- added execute_python
    input_guardrails=[tasha_guardrail],
    handoffs=[calculator_agent],
    model_settings=ModelSettings(temperature=0),
)

# ----------------------------
# Bedrock AgentCore app entry
# ----------------------------
app = BedrockAgentCoreApp()

@app.entrypoint
async def invoke(payload):
    user_message = payload.get("prompt", "Data, reverse the main deflector array!")
    output = ''
    try:
        result = await Runner.run(data_agent, user_message, session=session)
        output = result.final_output
    except InputGuardrailTripwireTriggered:
        output = "I'd really rather not talk about Tasha."
    return {"result": output}

if __name__ == "__main__":
    app.run()
```

Restricted Mode ⊗ 0 ⚠ 23

Build / test / deploy with AgentCore Starter Toolkit

```
pip install bedrock-agentcore-starter-toolkit
```

You can just run your .py file locally, and you get a service running on port 8080

- Hit it with POST and your json

To prepare for deployment:

- `agentcore configure -e <Python file containing the entry point>`
- The toolkit will guide you through additional info needed
- Explicitly enable observability in your AWS account (one-time thing)

Deploy

- `agentcore launch --env <Any env variables to include>`
- --local if you want to run locally (requires Docker / Finch / Podman)

Test

- Using `agentcore invoke <json>`
- Using the AWS SDK
- Using REST

Cleanup

- `agentcore destroy`

You can also deploy MCP servers!

No special integration at all in your code

Just package up your code, requirements.txt, and __init__.py

For authorization, you can use Cognito

- AgentCore will check for valid bearer tokens in requests

Let’s deploy our agent with AgentCore!

Playing with observability

WITH CLOUDWATCH GENAI OBSERVABILITY

Hang on, what are agents?

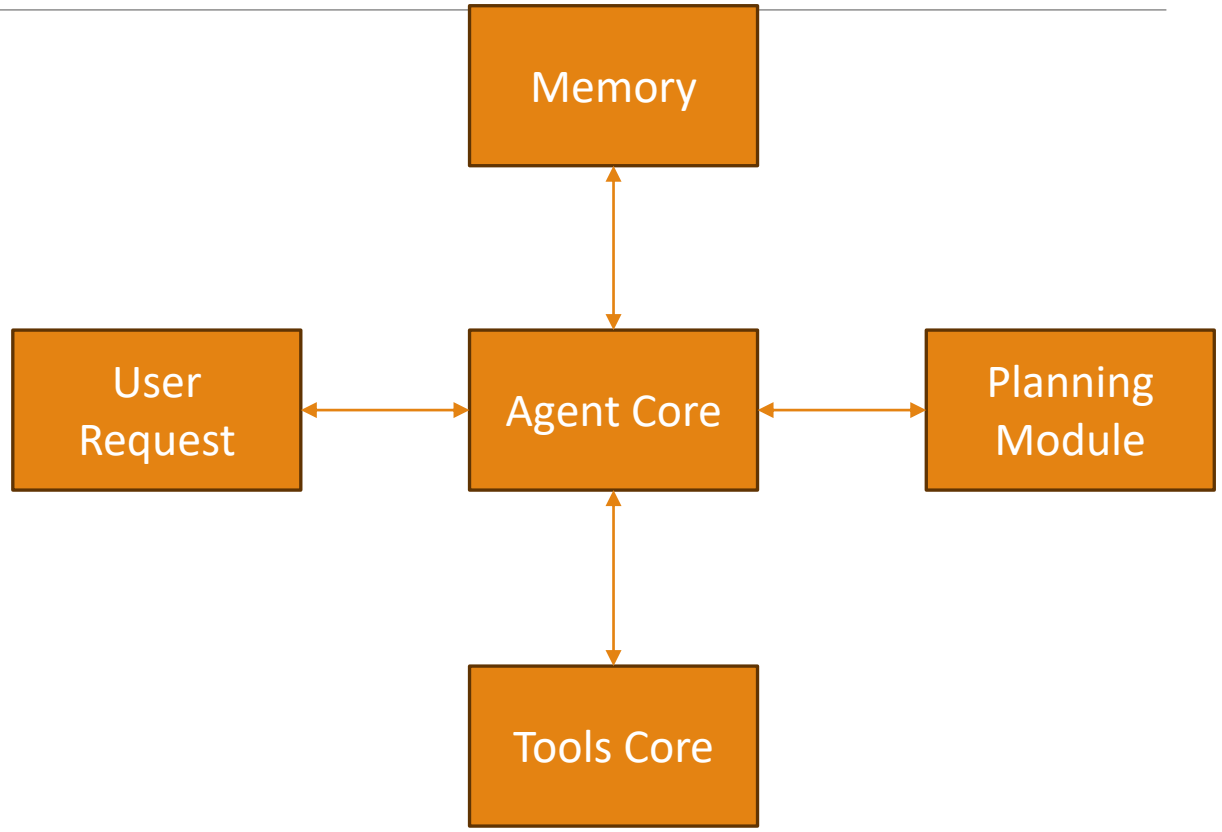
Giving tools to your LLM!

The LLM is given discretion on which tools to use for what purpose

The agent has a memory, an ability to plan how to answer a request, and tools it can use in the process.

In practice, the “memory” is just the chat history and external data stores, and the “planning module” is guidance given to the LLM on how to break down a question into sub-questions that the tools might be able to help with.

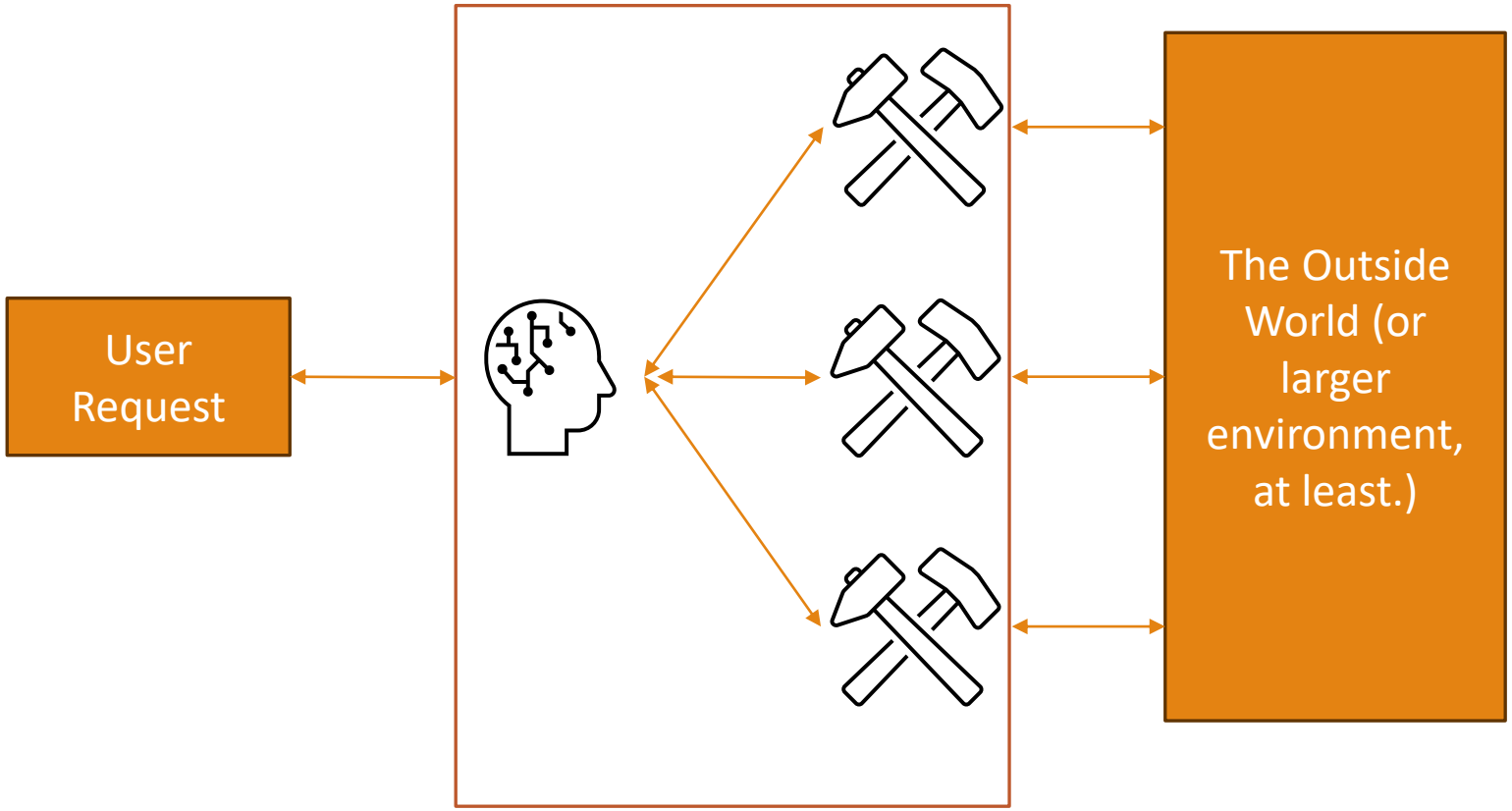
Prompts associated with each tool are used to guide it on how to use its tools.



Conceptual diagram of an LLM agent, as described by Nvidia (https://developer.nvidia.com/blog/introduction-to-llm-agents)

LLM Agents: A More Practical Approach

- “Tools” are just functions provided to the tools API.
- Prompts guide the LLM on how to use them.
- Tools may access outside information, retrievers, other Python modules, services, etc.



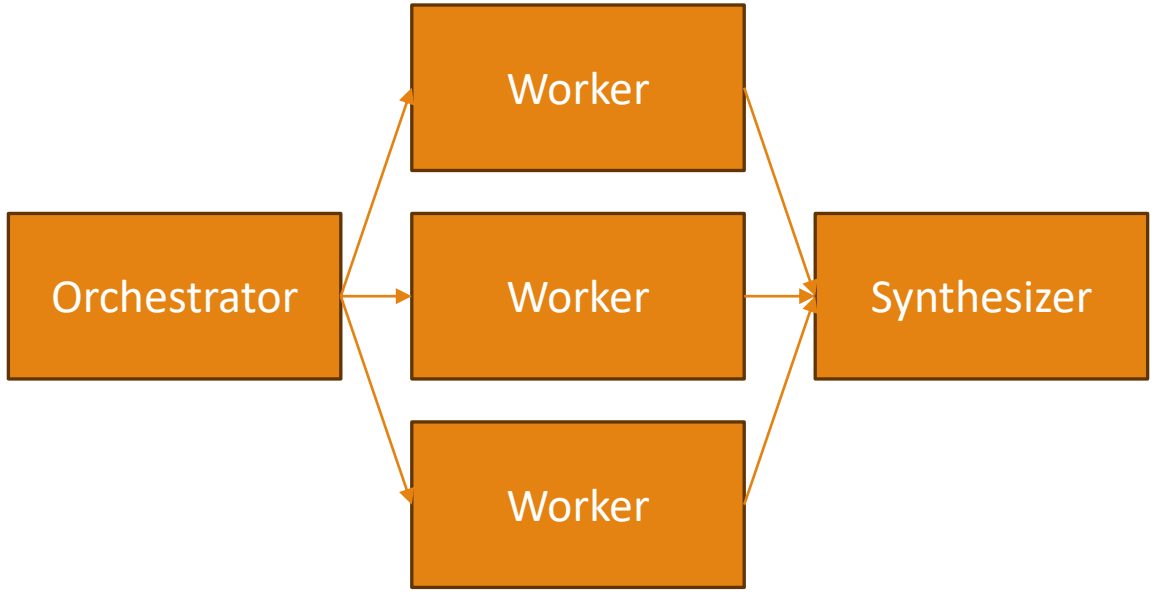
Multi-agent systems: an example

Similar to just giving tools to a single agent

- But many tools may be used at once
- For example, coding agents may need to operate on multiple files in different ways to achieve a larger task
- Workflows that require complex decision making can benefit from these agents (otherwise, keep it simple with deterministic workflows.)

The Orchestrator breaks down tasks and delegates them to worker LLM's

A Synthesizer can then combine their results



For example, this could be a translation tool being asked to translate to multiple languages at once.

Each LLM here may have its own tools and memory.

Adding authorization

Just set up a Cognito user pool for OAuth / JWT inbound authorization (IAM-based is present by default)

Specify the client ID(s) and discovery URL when running `agentcore configure`

Send a valid bearer token in the headers

No code changes needed!

Outbound authorization

- For calling external tools, MCP etc.
- Set up credential providers with `aws agent-credential-provider` CLI command
- Use `@requires_access_token` decorator on functions that access the external API

```
import asyncio
from bedrock_agentcore.identity.auth import requires_access_token,
requires_api_key


# This annotation helps agent developer to obtain access tokens from
external applications
@requires_access_token(
    provider_name="google-provider",
    scopes=["https://www.googleapis.com/auth/drive.metadata.readonly"]
, # Google OAuth2 scopes
    auth_flow="USER_FEDERATION", # 3LO flow
    on_auth_url=lambda x: print("Copy and paste this authorization url
to your browser", x), # prints authorization URL to console
    force_authentication=True,
)
async def read_from_google_drive(*, access_token: str):
    print(access_token) #You can see the access_token
    # Make API calls...
    main(access_token)


asyncio.run(read_from_google_drive(access_token=""))
```

Let’s add inbound auth to our agent

Adding AgentCore Memory

Short-term

- Chat history within a session / immediate context
- Enables conversations
- API centered around Session objects that contain Events

Long-term

- Stores “extracted insights”
- Summaries of past sessions
- Preferences (your coding style and favorite tools, for example)
- Facts you gave it in the past
- API involves “Memory Records” that store structured information derived from agent interactions
  - “Strategies” for user preferences, semantic facts, session summaries

This all needs to be stored somewhere!

- The OpenAI Agents SDK gives you a SQLLite implementation
- But maybe you need something that scales better, and is serverless
- Enter AgentCore Memory

This involves some coding.

You'll need to modify your agent code to integrate the AgentCore Memory API calls

You need to explicitly store, retrieve, and delete these memories

Naturally AWS's own Strands framework makes this pretty easy

Sample code is available for LangChain/LangGraph as well

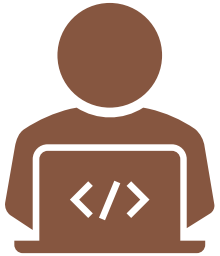
But for OpenAI, we're on our own

- OpenAI Agents use "Session" objects for memory
- They don't match up really well… but we can make it work…

AgentCore Built-In Tools

Browser Tool

Allows control of your browser to interact with the web



Code Interpreter

Lets you run code (in an isolated container)

Python, JavaScript, TypeScript

Importing Bedrock Agents

Bedrock has its own system for endpoints…

- But maybe you want to build on that or something.

Building agents in Bedrock is super easy

Importing is just a matter of running `agentcore import-agent`

- This generates Strands code (or LangChain / LangGraph) in an output directory
- From there you can test or deploy it like any other AgentCore agent

Let’s build an agent in Bedrock and import it.

LET’S PLAY WITH S3 VECTORS WHILE WE’RE AT IT!

AgentCore Gateway

Addresses the problem of using external tools at scale

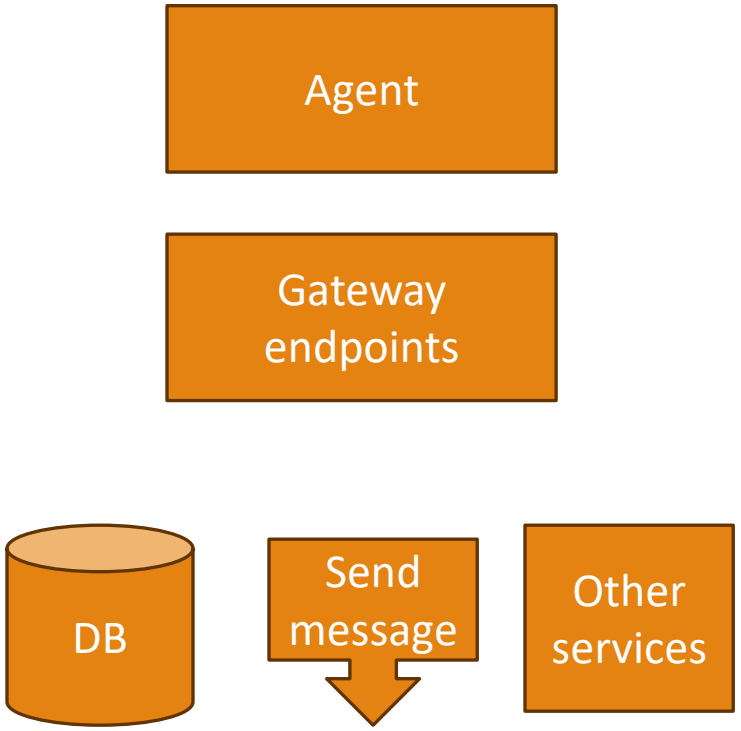
Convert APIs, Lambda functions, or other services into MCP tools

- Targets can be OpenAPI (REST), Smithy models (that's an AWS thing), or Lambda

Agents can then access them through Gateway endpoints

- Manages OAuth security / credentials

Semantic tool selection



AgentCore Identity

This is different from OAuth identity for users and connecting to services we talked about earlier

This is about your agent's identity / identities

Secure access to external tools and AWS services

Central repository for all of your agent identities

- Similar to a Cognito user pool

Secure credential storage

OAuth 2.0 support

- Built-in support for Google, GitHub, Slack, Salesforce, Atlassian

There is a lot of depth to this, but you probably don't need it right away

- Refer to https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/identity.html